EURECOM
Department of Digital Security
Campus SophiaTech
CS 50193
06904 Sophia Antipolis cedex
FRANCE

Research Report RR-19-342

SwaNN: Switching among Cryptographic Tools for Privacy-Preserving Neural Network Predictions

January 28th, 2019
Last update August 31th, 2019

Gamze Tillem, Beyza Bozdemir, and Melek Önen

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : G.Tillem@tudelft.nl, {Beyza.Bozdemir,Melek.Onen}@eurecom.fr

[1]EURECOM's research is partially supported by its industrial members: BMW Group Research and Technology, IABG, Monaco Telecom, Orange, Principauté de Monaco, SAP, Symantec.

SwaNN: Switching among Cryptographic Tools for Privacy-Preserving Neural Network Predictions

Gamze Tillem, Beyza Bozdemir, and Melek Önen

Abstract

The rise of cloud computing technology led to a paradigm shift in technological services that enabled enterprises to delegate their data analytics tasks to cloud servers which have domain-specific expertise and computational resources for the required analytics. Machine Learning as a Service (MLaaS) is one such service which provides the enterprises to perform machine learning tasks on a cloud platform. Despite the advantage of eliminating the need for computational resources and domain expertise, sharing potentially sensitive data with the cloud brings a privacy risk to the enterprises. In this paper, we propose SwaNN, a protocol to perform neural network predictions for MLaaS under privacy preservation. SwaNN brings together two well-known techniques for secure computation: partially homomorphic encryption (PHE) and secure two-party computation (2PC), and computes neural network predictions by switching between the two methods. The hybrid nature of SwaNN enables to maintain the accuracy of predictions and to optimize the computation time and bandwidth usage. Our experiments show that SwaNN achieves a good balance between computation and communication cost in neural network predictions compared to the state-of-the-art proposals.

Index Terms

privacy, neural networks, secure two-party computation, homomorphic encryption

Contents

List of Figures

1 Introduction

Neural networks are a method of supervised machine learning which aims to solve a classification problem. It computes the classification in two phases: a training phase in which a model is trained from previous observations whose classifications are known beforehand; a prediction phase in which a classification is computed for a new observation using the trained model [1].

Although the research on neural networks dates back to 1980s [2], they had not been commonly used due to their long training times. With the recent technological advances and the adaptation of GPUs in computation systems, the training time for neural networks is reduced significantly [3]. The improvement in performance triggered the popularity of neural networks, which in turn provided an outstanding success in certain fields such as image classification [3,4], face recognition [5], and board games [6].

The success of neural networks attracted many companies to apply it to their businesses. However, it is difficult for companies to successfully benefit from neural networks without having adequate computational resources and expertise in machine learning. Machine Learning as a Service (MLaaS) emerged as a solution to this problem. MLaaS enables the clients to outsource their machine learning tasks to a cloud platform which has computational resources and machine learning expertise [7]. A major risk that is challenging enterprises in using MLaaS is the sensitivity of the data sent to the cloud. The concern of exposing privacy-sensitive data in MLaaS services requires the design of privacy-preserving protocols for machine learning methods.

In this paper, we aim to design one such protocol for MLaaS to compute neural network predictions under privacy preservation. We assume that the network model has already been computed during a previous training phase, and we only focus on the privacy of data items during the prediction phase. Indeed, this problem drew the attention of researchers recently and several mechanisms that provide privacy protection in neural network predictions are proposed. The existing solutions that rely on cryptographic tools can be regrouped mainly in two categories. The solutions that are based on homomorphic encryption (HE) [1,8–10] enable computation of linear operations and low-degree polynomials non-interactively, where computations are performed by an external semi-trusted server. These solutions usually incur high computational cost due to the expensive nature of homomorphic encryption systems. Also, the restriction of linear and low-degree polynomial operations degrades the accuracy of prediction. Secure two-party computation (2PC)-based solutions [11–13], on the other hand, provide more realistic computation performance and seem better in maintaining the accuracy of predictions. However, the interactive nature of 2PC-based solutions leads to a higher bandwidth usage compared to HE-based alternatives.

Having studied existing solutions, we aim to take the best of both worlds and optimize the computational and the communication overhead at the same time. We propose a hybrid protocol, **SwaNN**, which switches the computations between

HE and 2PC. Instead of using leveled or somewhat homomorphic encryption, we make use of partially homomorphic encryption (more specifically the additively homomorphic Paillier encryption) to perform linear operations over encrypted data. This also helps the solution reduce the computational cost. Non-linear operations are supported thanks to the use of 2PC. We show how to easily switch from one cryptographic tool to the other. The combination of these two cryptographic tools helps maintain the accuracy of predictions. The idea of using a hybrid protocol for private neural network predictions is proposed in Gazelle [14] as well, which combines Yao's garbled circuits with a dedicated lattice-based additively homomorphic encryption scheme. Our proposal differs from Gazelle by using well-known simple cryptographic tools, which make the adoption of our proposal more practical.

SwaNN is designed to support two different settings: a client-server setting and a non-colluding two-server setting. In the client-server setting, the majority of operations are delegated to the server, and the client helps the server in intermediate steps. In the two-server setting, the servers are provided the data beforehand, and they perform the computations simultaneously, with a balanced workload on both servers. Our contributions can be summarized as follows:

- We propose a hybrid protocol for neural network predictions, which is based on the additively homomorphic Paillier encryption scheme and secure two-party computation. We show how each underlying operation can be supported easily with the use of these two schemes only.
- Our protocol is flexible since it is suitable both for the client-server setting and the non-colluding two-server setting.
- Compared to existing works, our protocol proposes several optimizations for the computations in the linear layers of neural networks which improves the efficiency in terms of computation cost. These optimisations consist of some data packing dedicated to the Paillier cryptosystem and the use of multi-exponentiation algorithm to reduce the cost of multiplications.
- The empirical results show that our protocol can compute the prediction on a neural network with two activation layers in 10 seconds with 1.73 MB bandwidth usage which is 30-fold better in computation cost than the state-of-the-art HE-based solution and 27-fold more efficient in bandwidth usage than the state-of-the-art 2PC-based solution.

Section 2 introduces neural networks and the underlying cryptographic tools used in SwaNN. In Section 3, we discuss the contribution of our work along with the prior work. We describe our protocol in Section 4. In Section 5, we present the empirical evaluation of our work. We conclude our paper in Section 6.

2 Preliminaries

In this section, we present the necessary background information on convolutional neural networks and the cryptographic primitives we use in our design. Table 1 summarizes the notation we use throughout the paper.

Table 1: Notation table.

Symbol	Explanation
$\mathbf{X} = \{x_{0,0}, x_{0,1}, \ldots\}$	Input matrix
$\mathbf{Y} = \{y_{0,0}, y_{0,1}, \ldots\}$	Output matrix
$\mathbf{W} = \{w_{0,0}, w_{0,1}, \ldots\}$	Model weight matrix
$\mathbf{B} = \{b_{0,0}, b_{0,1}, \ldots\}$	Bias matrix
$f(\mathbf{X}, \mathbf{W}) = \mathbf{Y}$	Function f that operates on inputs $\mathbf{X}, \mathbf{W}$ and returns output $\mathbf{Y}$
N	Plaintext modulus of the Paillier cryptosystem
N^2	Ciphertext modulus of the Paillier cryptosystem
ℓ	Bit size for 2PC operations
κ	Security parameter
$a \in_R A$	a is chosen uniformly randomly from A
$\odot$	Dot product symbol
$\otimes$	Matrix multiplication symbol
$[\cdot]$	Paillier ciphertext
$\langle \cdot \rangle_i$	Input share of party i for 2PC operations

2.1 Convolutional Neural Networks

Convolutional neural networks are specifically designed for image recognition. They combine a series of layers to perform classification. Each layer takes an input $\mathbf{X}$, evaluates a function f on the input along with a weight matrix $\mathbf{W}$, and returns an output $\mathbf{Y}$ to the subsequent layer. Fig. 1 illustrates an overview of the neural network structure. The first layer of the neural network is the input layer, where the input is provided to the network. The last layer is the output layer, where the result of the classification is revealed. The layers between input and output are called hidden layers. Each hidden layer evaluates the function associated with that layer on its input and delivers the output to the next layer.

Below we describe the most common hidden layers used in convolutional neural networks.

Convolutional Layer (`Conv`) is based on convolutional filtering in image processing. It applies a filter, $\mathbf{W}$, on the input $\mathbf{X}$ by sliding the filter every time to work on each index of the input matrix. The sliding factor is called stride (s). The size of the input can be adjusted to fit to filter size by appending some border values

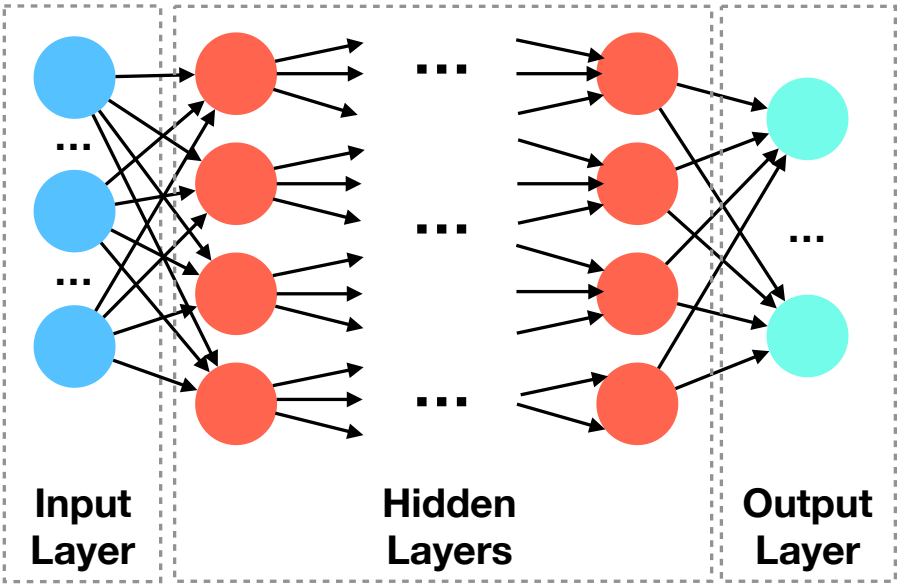

Figure 1: An overview of the neural network structure with input, output, and hidden layers.

which is called padding (p). The operation performed in convolutional layer is a dot product such that

$$f(\mathbf{X}, \mathbf{W}) = \mathbf{W} \odot \mathbf{X} + \mathbf{B} \tag{1}$$

$$= \sum w_{i,j} \times x_{i,j} + b_{i,j}. \tag{2}$$

Fig. 2 shows the operation of convolutional filter on the input.

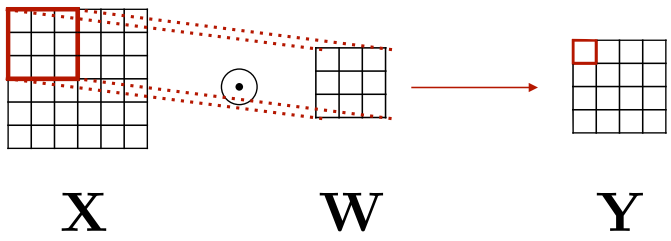

Figure 2: Convolutional filtering in convolutional layer.

Fully Connected Layer (FC) connects each neuron in the current layer to each neuron in the previous layer along with a weight value. The operation is a matrix multiplication

$$\mathbf{X}^t = \mathbf{X}^{t-1} \otimes \mathbf{W}, \tag{3}$$

where $\mathbf{X}^t$ are the neurons of the current layer and $\mathbf{X}^{t-1}$ are the neurons of the previous layer.

Pooling Layer (`Pool`) is a scaling layer which reduces the size of the input matrix. Reduction is performed by sliding a filter on the input and performing the pooling operation on each area that is covered by the filter. The two common types of pooling are

- *max pooling* where the maximum value within the area covered by the filter is selected.
- *average pooling* where the average of the values within the area covered by the filter is selected.

Unlike previous layers, the pooling operations are nonlinear.
Activation Layer (`Act`) is also a nonlinear layer. In this layer, a nonlinear activation function $f(x_{i,j}) = y_{i,j}$ is applied to each neuron of the input layer. The size of the output is same with the size of the input. Most commonly used activation functions are

$$\textit{Sigmoid: } \frac{1}{1+e^{-x_{i,j}}}, \tag{4}$$

$$\textit{Hyperbolic tangent (tanh): } \frac{e^{2x_{i,j}}-1}{e^{2x_{i,j}}+1}, \tag{5}$$

$$\textit{Rectified Linear Units (ReLU): } \max(0, x_{i,j}). \tag{6}$$

2.2 Homomorphic Encryption

Homomorphic property enables a cryptosystem to perform operations on the encrypted input without decryption. If a cryptosystem enables both additions and multiplications under encryption, it is called fully homomorphic whereas if it supports a single type of operation, it is called partially homomorphic. Despite their flexibility on performing both types of operations, fully homomorphic cryptosystems are expensive in computation. In contrast, partially homomorphic schemes are more efficient with their reasonable computation cost.

In this paper, we use a partially homomorphic cryptosystem, namely the Paillier cryptosystem [15] which supports additive homomorphism. The public key of the Paillier cryptosystem is (N, g), where N is the product of two large primes p and q, and $g \in \mathbb{Z}^*_{N^2}$. The private key is (λ, μ), where $\lambda = lcm(p-1, q-1)$ and $\mu = (L(g^\lambda \mod N^2))^{-1} \mod N$. The encryption function of the Paillier cryptosystem is probabilistic such that every encryption of the same plaintext results in a different ciphertext. Encryption of a message $m \in \mathbb{Z}_N$ on modulus N is computed as $E(m) = g^m \cdot r^N \mod N^2$, where $r \in \mathbb{Z}^*_{N^2}$. An encrypted message $E(m)$ can be decrypted using the formula $m = L(c^\lambda \mod N^2) \cdot \mu \mod N$. As it is described in [16], the decryption function of the Paillier cryptosystem supports threshold decryption. In our paper, when necessary, we use a 2-out-of-2 variant of the threshold decryption which distributes the private key among two parties. Successful decryption requires both parties to compute the decryption function.

Using the Paillier cryptosystem, it is possible to compute addition and scalar multiplication on encrypted messages as depicted in the following equations

$$\begin{aligned} E(m_1) \times E(m_2) &= g^{m_1} \cdot r_1^N \times g^{m_2} \cdot r_2^N \mod N^2 \\ &= g^{m_1+m_2} \cdot (r_1 \cdot r_2)^N \mod N^2 \\ &= E(m_1 + m_2), \end{aligned} \tag{7}$$

$$\begin{aligned} E(m)^c &= g^{c\dot{m}} \cdot (r^c)^N \mod N^2 \\ &= E(c \cdot m). \end{aligned} \tag{8}$$

We refer readers to [15] for the details of the cryptosystem. In the rest of the paper, we represent a Paillier ciphertext with $[\cdot]$ symbol.

2.3 Secure Two-party Computation

Secure two-party computation enables two parties to jointly compute a function f on their inputs without revealing the inputs to each other. In our work, we use two different methods for secure two-party computation which are arithmetic secret sharing [17] and Boolean secret sharing [18].

Arithmetic secret sharing: Given two parties P_1, P_2, their arithmetic shares on an ℓ-bit value m are $\langle m \rangle_1$ and $\langle m \rangle_2$ such that $\langle m \rangle_1 + \langle m \rangle_2 \equiv m \mod 2^\ell$. Arithmetic sharing enables computation of addition and multiplication operations on secretly shared values. The addition of two secret shared values is computed locally by each party as $\langle z \rangle_i = \langle x \rangle_i + \langle y \rangle_i$. Computing multiplication is, however, more complicated. It can be computed using Beaver's multiplication triplet technique [17]. We refer readers to [17] for a detailed explanation of the technique.

Boolean secret sharing: Given the parties P_1, P_2, their Boolean shares on a single bit value m are $\langle m \rangle_1$ and $\langle m \rangle_2$ such that $\langle m \rangle_1 \oplus \langle m \rangle_2 \equiv m \mod 2$, where $\oplus$ is the XOR operation. Boolean circuits can compute both linear and nonlinear operations.

In arithmetic sharing, additions can be computed locally without any additional cost. A multiplication operation requires some additional computation and communication cost; however, it is less expensive than the multiplication in Boolean sharing [19]. Therefore, in our protocol, we use arithmetic sharing for addition and multiplication operations. When other types of operations such as comparisons are needed, we use Boolean sharing.

3 Prior Work

In this section, we regroup existing privacy preserving neural networks into several categories based on the underlying privacy enhancing technology. We further highlight their relevance with respect to our protocol.

The first category of solutions consists of solutions based on secure multi-party computation. In [11], SecureML designs a privacy-preserving neural network training and classification method using 2PC, where clients secretly share their own private data among two non-colluding servers. SecureML builds the model with the stochastic gradient descent method. Authors compute ReLU using garbled circuits and implement polynomial approximations of nonlinear functions such as the sigmoid and softmax functions. Additionally, a solution for switching between arithmetic and Yao's sharing is proposed. As an extension to SecureML, authors [12] propose ABY^3 which shares the private inputs between three non-colluding servers. To securely share sensitive data among three servers, the authors redefine arithmetic, Boolean and Yao's sharings of the ABY framework [19]. MiniONN, proposed by Liu et al. [13], also uses 2PC for privacy-preserving neural network operations. Different from [11] and [12], MiniONN focuses on the prediction phase only. The authors propose a 2PC protocol between the client and the cloud. The client and the cloud additively share each of their input and output values for each layer of the neural network. To ensure data privacy, MiniONN defines oblivious transformations for each CNN operation and implements the transformations using the ABY framework. Furthermore, Rouhani et al. [20] propose DeepSecure which is based on Yao's garbled circuits to securely compute the deep learning model. The authors are able to use sigmoid and tanh as activation functions thanks to the optimization of garbled circuits. Another study which uses 2PC is Chameleon by Riazi et al. [21]. Authors propose a protocol that switches among sharing circuits for secure function evaluation, where two parties jointly perform a computation without disclosing their inputs. Chameleon can be considered as an alternative protocol to ABY [19]. TFEncrypted [22] is another framework which enables secure computation in TensorFlow [23] using secret sharing and secure channels between the parties. Moreover, a very recent scheme named SecureNN [24] uses secure three-party computation for the training and classification phases with convolutional neural networks using the MNIST dataset. SecureNN shares the input and output among two parties using 2-out-of-2 arithmetic shares, and the third party joins the protocols during the online computation. In comparison to SecureNN, SwaNN requires interaction with the client for the computation of the square function only, which is less than compared to the interactions of three parties in SecureNN.

In the second category, we analyze fully homomorphic encryption (FHE)-based solutions. To the best of our knowledge, CryptoNets [8] is the first privacy-preserving neural network protocol which is based on FHE. Authors in [8] use the SEAL library [25] to compute convolutional neural network predictions on encrypted images. Similar to CryptoNets, CryptoDL [26], Chabanne et al. [1] and Ibarrondo et al. [27] use FHE for privacy-preserving neural networks. The main difference with CryptoNets is the fact that they approximate nonlinear functions with higher degree polynomials using different techniques such as Taylor series, numerical methods or Chebyshev polynomials. The use of batch normalization is also proposed to obtain some performance gain. The goal of all these solu-

tions is to keep a good level of accuracy while using FHE to protect the input data. Later on, Bourse et al. [28] uses a conversion of a trained neural network to a Discretized Neural Network (DiNN) using an efficient FHE called TFHE [29]. Authors claim that DiNN can be used for deep neural networks with large number of neurons. Similarly, TAPAS [30] also proposes binary neural networks over TFHE-encrypted data. However, TAPAS differs from [28] mainly due to the ability of the server to update the neural network at any time without the need for the data being re-encrypted by the client. More recent works, namely [31] and [32] propose the idea of training neural networks over FHE-encrypted data and classifying encrypted predictions. In their studies, the client supplies the training data in its encrypted form using its own public key, and the server trains this encrypted data to build the encrypted model. This model is further used by its owner to classify a new encrypted input. Because both the training data and the model are encrypted, the server cannot discover any information on both phases. Authors claim to achieve a reasonable performance. Moreover, Faster CryptoNets [33] is a system employing the sparse encodings over the neural network model and data when it remains encrypted under the FHE scheme. The authors propose some efficient polynomial approximations for activation functions. The scheme also includes a training phase that uses differential privacy to protect the data.

In comparison with existing solutions from these two categories, we propose a hybrid protocol that combines 2PC with partially HE. Our goal in SwaNN is to come up with private neural network predictions by making use of more simple cryptographic tools, where the client can obtain the prediction result without disclosing its input to the server, and the privacy of server's neural network against the client is ensured. Therefore, we propose to take advantage of both privacy enhancing technologies and optimize their respective costs (computational and/or communication cost). To reduce the computational cost, FHE is replaced with the additively homomorphic Paillier encryption scheme. This algorithm is used to compute linear operations and the x^2 function. Additionally, we obtain better performance results for computing nonlinear operations thanks to the use of 2PC.

Few early approaches, such as [9] and [10], also use the Paillier encryption scheme and Yao's garbled circuits. Although these solutions seem similar to our proposal, they study very small neural networks and suffer from significant communication overhead due to frequent client-server interactions. Furthermore, authors in [9] and [10] do not provide any performance results of their solution executed over the encrypted data. Additionally, Gazelle [14] is a secure neural network inference scheme implemented under a dedicated lattice-based additively homomorphic encryption scheme proposed in the paper. This solution also makes use of Yao's garbled circuits to perform ReLU and to reduce the noise in the ciphertext. Unfortunately, this results in linear growth in computation and communication costs, and it also increases the depth of the circuit. Instead, we make use of a secure square protocol to compute the activation function without Yao's garbled circuits.

Lastly, there exist several works, such as [34], [35], and [36], which propose

to combine some machine learning techniques, including neural networks, with trusted hardware.

4 SwaNN

In the Machine Learning as a Service (MLaaS) model, the client has limited computation capabilities or knowledge of machine learning. Thus, he outsources the computations to the server who has expertise in performing machine learning with adequate computation power. In a desirable scenario, the workload on the client side should be minimized. In this paper, we consider two different scenarios both of which aim to minimize the computations at the client side and the overall computation cost while maintaining privacy.

1^{st} Scenario - Client-Server: In this scenario, a client shares a private image with a server. The server, which holds the neural network model, computes the prediction result on the private image. The majority of the computations are performed by the server. The client helps the server perform decryptions and/or circuit evaluations when it is necessary.

2^{nd} Scenario - Two-Server: To reduce the workload on the client side further, we design a two-server setting where two semi-honest non-colluding servers perform the computations together. The client provides the servers their shares on the input and private keys. Thus, the computations on the client side are completely delegated to the servers. In such a setting to fully utilize the capabilities of both servers, one image can be provided to each server such that at one execution they evaluate two images simultaneously.

In both scenarios, we assume a semi-honest security model, where the parties do not collude. In this security model, parties exactly follow the protocol steps. However, they are curious to obtain some information from their output and intermediary messages. In both of our scenarios, the client's goal is to hide the image content and the result of classification from the server. On the other hand, the server does not want to reveal the model parameters used during computations to the client. In the rest, we explain the computation of private neural network predictions for both scenarios individually.

4.1 Scenario 1: Client - Server

In the client-server scenario, the majority of computations are performed by the server, and the client is involved when intermediary decryptions are needed. Figure 3 illustrates our first scenario. The client encrypts an image with his public key and sends it to the server who computes the secret prediction result using the neural network parameters. Depending on the operation performed by the server, the client might involve in the computations.

In Section 2.1, we summarize the common layers for convolutional neural networks and the necessary operations to compute the functions in these layers. Be-

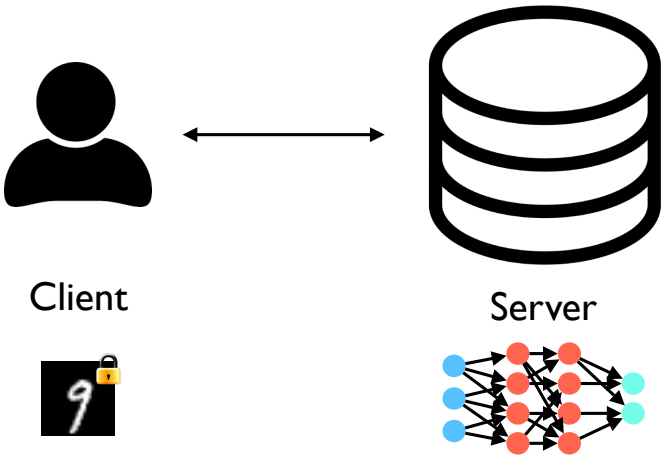

Figure 3: Client-server scenario for SwaNN with a single input image.

low we explain how we can compute these layers under privacy preservation in the client-server scenario. Essentially, we separate the computations into two phases as non-interactive phase and interactive phase. In the non-interactive phase, the operations are performed by the server without the client's involvement. The interactive phase, however, requires the collaboration of the server and the client for computations. By convention, convolutional neural networks start with a convolutional layer. Therefore, we assume that the computations always start with an image that is encrypted under the Paillier cryptosystem by the client. This encrypted image is sent to the server.

4.1.1 Non-interactive phase

In this phase, the server, who has received the encrypted image, computes the linear layers of the neural network as follows.

Convolutional Layer: The main operation in the convolutional layer is the dot product. Given an input image $\mathbf{X}$ and a weight matrix $\mathbf{W}$, their dot product is computed as $\mathbf{Y} = \sum x_{i,j} \times w_{i,j}$. When the input image is encrypted with the Paillier cryptosystem and the weight matrix is in plaintext, using the homomorphic property of encryption, the dot product is computed as

$$[\mathbf{Y}] = \left[\sum x_{i,j} \times w_{i,j}\right] = \prod [x_{i,j}]^{w_{i,j}}. \tag{9}$$

Since this computation does not require any decryption, it can be performed non-interactively by the server.

Fully Connected Layer: Fully connected layer requires to compute a matrix multiplication. The underlying operation for matrix multiplication is the dot product, but it has to be performed for each column and row pair. Given an encrypted image as input, the fully connected layer is computed by performing Equation 9 repetitively.

Mean Pool Layer: Despite the computation of pooling layer is nonlinear, following the convention in the state-of-the-art works [8, 13] we use a linear approximation of the mean pooling operation. Originally, the computation of mean pooling requires the summation of the values within a subgroup and then a division by the subgroup size. Following the approach in [8, 13], we compute scaled mean pool instead of the mean pool, where the summation is performed, but the division is omitted. The scaled mean pool can be computed by additive homomorphic property of the Paillier cryptosystem without interaction.

4.1.2 Interactive phase

In this phase, the server computes the nonlinear layers of the neural network in collaboration with the client as follows.

Activation Layer: Computing the nonlinear activation function in neural networks is a challenging task when privacy preservation is required. Since the Paillier homomorphic encryption supports only additions, activation functions cannot be computed without performing decryption. In the existing literature on privacy-preserving neural networks, there are two approaches to compute the activation function.

The first approach is to compute a polynomial approximation of the function. CryptoNets [8] and MiniONN [13] use x^2 as the approximation of the sigmoid function. In SwaNN, we propose two solutions to compute the approximation function x^2. Since the Pailler cryptosystem does not support multiplications, as a first solution, we design an interactive secure square function using the additively homomorphic property of the Paillier cryptosystem. Our solution adapts the secure multiplication protocol in [37] to a secure square protocol (see Protocol 1). Our second solution for the computation of x^2 uses a multiplication operation under arithmetic sharing. The multiplication requires to switch the computations from homomorphic encryption to arithmetic sharing. Later in this section, we explain how we can perform such a switching operation.

The second approach to compute the ReLU activation function using secure two-party computation techniques. MiniONN [13] and SecureML [11] are the state-of-the-art solutions which use arithmetic circuits and Yao's garbled circuits [38] to compute the ReLU activation function. In SwaNN, we adapt a similar approach and use the circuit-based approach when the computation of ReLU is required. We compute ReLU using a comparison gate under Boolean sharing.

Max Pool Layer: Unlike the mean pool layer, we do not use an approximation function for the computation of the max pool layer. Instead, we implement the maximum pooling using the comparison gates under Boolean sharing. We perform the max pool layer right after the activation layer to reduce the number of switching operations between 2PC and PHE.

Switching between HE and 2PC In the previous subsections, we describe how to compute linear and nonlinear layers of neural networks using partially homo-

Protocol 1: Secure Square Protocol

Client (pk, sk)		Server (pk)
		$[x], r \in_R \{0,1\}^{\ell+\kappa}$
		$[x_r] \leftarrow [x] \cdot [r]$
		$[x_r] \leftarrow [x + r]$
	$\xleftarrow{[x_r]}$	
$x_r \leftarrow \text{decr}([x_r])$		
$x_r^2 \leftarrow x_r \cdot x_r$		
$[x_r^2] \leftarrow \text{enc}(x_r^2)$		
	$\xrightarrow{[x_r^2]}$	
		$[x_r^2] \cdot \left([r^2] \cdot [x]^{2r}\right)^{-1}$
		$[x^2] \leftarrow \left[x_r^2 - r^2 - 2xr\right]$

morphic encryption (PHE) and secure two-party computation (2PC). Since linear and nonlinear operations follow each other repetitively, we need a secure switching mechanism between the two cryptographic techniques. We design a protocol for secure switching which is similar to the secure decryption mechanism described in [39]. Protocol 2 and 3 demonstrate the steps of switching from PHE to 2PC and 2PC to PHE, respectively.

Protocol 2: PHE to 2PC Secure Switching Protocol

Client (pk, sk)		Server (pk)
		$[x], r \in_R \{0,1\}^{\ell+\kappa}$
		$[x + r] \leftarrow [x] \cdot [r]$
	$\xleftarrow{[x+r]}$	
$x + r \leftarrow \text{decr}([x + r])$		
$x + r \rightarrow \langle x + r \rangle_c + \langle x + r \rangle_s$		
	$\xrightarrow{\langle x+r \rangle_s}$	
		$r \rightarrow \langle r \rangle_c + \langle r \rangle_s$
	$\xleftarrow{\langle r \rangle_c}$	
$\langle x \rangle_c \leftarrow \langle x + r \rangle_c - \langle r \rangle_c$		$\langle x \rangle_s \leftarrow \langle x + r \rangle_s - \langle r \rangle_s$

Switching from PHE to 2PC (Protocol 2) requires to perform a secure decryption by masking the encrypted value with a random r. Once the client securely decrypts the masked value $x + r$, he creates the secret shares of it for himself and for the server as $\langle x + r \rangle_c$ and $\langle x + r \rangle_s$. In the mean time, the server creates the secret shares of the random r as $\langle r \rangle_c$ and $\langle r \rangle_s$ to remove the mask from the original value x. Finally, both parties perform a local subtraction on their shares $\langle x + r \rangle$ and $\langle r \rangle$ to compute the secret shared value $\langle x \rangle$ which is going to be used in 2PC computations.

Switching from 2PC to PHE (Protocol 3) reverses the former procedure. It

Protocol 3: 2PC to PHE Secure Switching Protocol

Client (pk, sk)		Server (pk)
$\langle x \rangle_c$		$\langle x \rangle_s$, $r' \in_R \{0,1\}^{\ell+\kappa}$
		$r' \rightarrow \langle r' \rangle_c + \langle r' \rangle_s$
	$\xleftarrow{\langle r' \rangle_c}$	
$\langle x + r' \rangle_c \leftarrow \langle x \rangle_c + \langle r' \rangle_c$		
		$\langle x + r' \rangle_s \leftarrow \langle x \rangle_s + \langle r' \rangle_s$
	$\xleftarrow{\langle x+r' \rangle_s}$	
$x + r' \leftarrow \langle x + r' \rangle_c + \langle x + r' \rangle_s$		
$[x + r'] \leftarrow enc(x + r')$		
	$\xrightarrow{[x+r']}$	
		$[x] \leftarrow [x + r'] \cdot [r']^{-1}$

starts with a secret shared value $\langle x \rangle$. Similar to the previous protocol, to prevent the leakage of the original value the parties reveal it after masking. Thus, the server generates a random mask r' and sends a secret share of the random $\langle r' \rangle_c$ to the client. Both parties perform an addition operation to mask $\langle x \rangle$, and then the server sends the masked value $\langle x + r' \rangle_s$ to the client. Client reveals $x + r'$ by adding the two shares and encrypts it with his public key. In the final step, the server removes the random mask from $[x + r']$ with a homomorphic subtraction.

4.2 Scenario 2: Two-Server

The client-server scenario necessitates a certain level of computation power from the client, despite the majority of the operations are performed by the server. To reduce the workload from the client's side, we design a second scenario which outsources the computations to two non-colluding servers. In this scenario, the client provides the input to both servers, and the servers perform the operations and return the result to the client. However, if only a single image is provided to the servers one of the servers is going to be idle during the non-interactive phase of the computations. Thus, we propose to provide one different image to each server to fully utilize the computation capabilities of the servers and classify two images at once.

Fig. 4 illustrates our scenario. The client encrypts two images with his public key and provides one image to each server. Furthermore, he creates shares of the private key for each server as described in [16] and sends the shares to each server. Similar to the first scenario, we divide the computations into two phases as non-interactive and interactive phases. In the non-interactive phase, the servers compute the linear operations on their inputs as the same way described in Section 4.1.1. The interactive phase and the switching phase are also similar to the description in Section 4.1.2, but they differ in the decryption procedure. In the first scenario, the client is responsible for performing the decryption operations. However, in

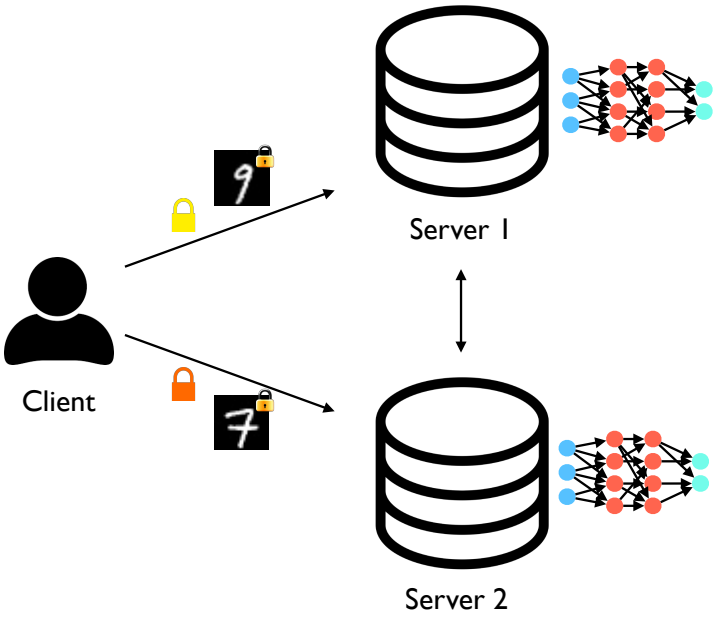

Figure 4: Two-server scenario for SwaNN with two input images.

the two-server scenario, the decryption task is also delegated to the servers along with their shares on the secret key. Therefore, the decryption function $\text{decr}([\cdot])$ in Protocol 1 and Protocol 2 is performed by both servers. To clarify the procedure, in Protocol 4 we illustrate how secure square protocol works when the computations are delegated to the two servers.

While the execution of non-interactive phase can be done by each server locally, the interactive phase requires the involvement of both parties. The servers can execute this phase sequentially based on a predetermined order, or they can execute it in parallel which improves the computation cost further.

4.3 Security Analysis

SwaNN aims to compute neural network predictions under the privacy preservation assumption in the semi-honest adversarial model. We assume the semi-honest adversary is non-adaptive and computationally bounded. In this security model, for both of the scenarios we propose, the two parties should not be able to retrieve any additional information from the protocol execution apart from their inputs, outputs, and intermediary messages. We achieve our security goal thanks to the security of the cryptographic techniques we use in the design of SwaNN. Both the Paillier cryptosystem and secure two-party computation are proven to be secure. In the non-interactive phase of SwaNN, the security is guaranteed by the semantic security of the Paillier cryptosystem. The Paillier cryptosystem satisfies semantic security against chosen plaintext attacks under decisional composite

Protocol 4: Secure Square Protocol in the two-server scenario

Server 1 (pk, sk_1)		Server 2 (pk, sk_2)
		$[x], r \in_R \{0,1\}^{\ell+\kappa}$
		$[x_r] \leftarrow [x] \cdot [r]$
		$[x_r] \leftarrow [x + r]$
		$[x_r]' \leftarrow \text{decr}_2([x_r])$
	$\xleftarrow{[x_r]'}$	
$x_r \leftarrow \text{decr}_1([x_r]')$		
$x_r^2 \leftarrow x_r \cdot x_r$		
$[x_r^2] \leftarrow \text{enc}(x_r^2)$		
	$\xrightarrow{[x_r^2]}$	
		$[x_r^2] \cdot \left([r^2] \cdot [x]^{2r}\right)^{-1}$
		$[x^2] \leftarrow \left[x_r^2 - r^2 - 2xr\right]$

residuosity assumption [15]. Thus, in the computation of convolutional, fully connected, and mean pool layers, the server(s) cannot reveal any valuable information from the encrypted messages on the condition that the encryption is performed with a key that meets the current security requirements.

The activation and max pool layers, on the other hand, requires interactive protocols between two parties during which the computations might be switched from homomorphic encryption to secure two-party computation, and vice-versa. Besides the security of the Paillier cryptosystem, arithmetic secret sharing and Boolean secret sharing, which are used in the interactive phase of SwaNN as secure two-party computation techniques, achieve indistinguishability given that the shares are generated from a uniformly random distribution [18]. Assuming that the Paillier cryptosystem and secure two-party computation are secure, the security of the interactive phase of SwaNN can be deduced to the security of switching or decryption operations. In the rest of this section, we provide a formal security proof using the simulation paradigm [40] to show that the switching and decryption operations can be performed securely. Due to limited space, we provide the proof only for Protocol 2, which switches the operations from homomorphic encryption to secure two-party computation in the client-server scenario.

In the simulation paradigm, the ideal security setting is outsourcing inputs of both parties to a trusted third party who can perform the computations and return the output. In the real-world setting, the security goal is to show that if an adversary $\mathcal{A}$ can attack the protocol in the real world, then the attack can be also performed by an adversary $\mathcal{S}$ in the ideal world. Since the attacks of $\mathcal{S}$ are not successful in the ideal setting, the attacks in the real world also fail and the protocol is proved to be secure in the real world. In Definition 41 and Definition 42, we provide the formal definitions for security and indistinguishability from [40].

Definition 41 (Computational Indistinguishability)**.** Let $X(a,\kappa)$ and $Y(a,\kappa)$ are two probability ensembles where $a \in \{0,1\}^*$ is the input of the parties and κ is the security parameter. $X(a,\kappa)$ and $Y(a,\kappa)$ are *computationally indistinguishable* (i.e. $X(a,\kappa) \stackrel{c}{\equiv} Y(a,\kappa)$) if there exists a negligible function $\mu(\kappa)$ for every nonuniform polynomial time algorithm D, and for every $a \in \{0,1\}^*$ and $\kappa \in \mathcal{K}$ such that

$$|\Pr[D(X(a,\kappa)) = 1] - \Pr[D(Y(a,\kappa)) = 1]| \leq \mu(\kappa). \tag{10}$$

Definition 42 (Definition of Security)**.** P_1 and P_2 are two parties who want to run a protocol π on their inputs x and y to compute a functionality $f(x,y)$ which outputs $f_1(x,y)$ and $f_2(x,y)$ for each party. In the execution of π, the view of parties are

$$\textbf{view}_1^\pi(x,y,\kappa) = (x, r_1; m_1, m_2, \cdots, m_t), \tag{11}$$
$$\textbf{view}_2^\pi(x,y,\kappa) = (y, r_2; m_1, m_2, \cdots, m_t), \tag{12}$$

where r_1, r_2 are the randomness of the parties, κ is the security parameter and m_i's are the intermediary messages received by each party. The output of π is $\textbf{output}^\pi(x,y,\kappa) = (\textbf{output}_1^\pi(x,y,\kappa), \textbf{output}_2^\pi(x,y,\kappa))$, such that $\textbf{output}_1^\pi(x,y,\kappa)$ and $\textbf{output}_2^\pi(x,y,\kappa)$ are the local outputs of P_1 and P_2. We say that π securely computes $f(x,y)$ in the presence of semi-honest, non-adaptive, computationally bounded adversaries, if there exist probabilistic polynomial-time simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\{\mathcal{S}_1(1^\kappa, x, f_1(x,y)), f(x,y)\} \stackrel{c}{\equiv} \{\textbf{view}_1^\pi(x,y,\kappa),\ \textbf{output}^\pi(x,y,\kappa)\}, \tag{13}$$
$$\{\mathcal{S}_2(1^\kappa, y, f_2(x,y)), f(x,y)\} \stackrel{c}{\equiv} \{\textbf{view}_2^\pi(x,y,\kappa),\ \textbf{output}^\pi(x,y,\kappa)\}. \tag{14}$$

Accordingly, Protocol 2 is a protocol π between a server and a client which computes the functionality f that switches the computations from PHE to 2PC. The client does not provide an input for π (i.e. his input is an empty string $\perp$) apart from the auxiliary inputs encryption and decryption keys (pk, sk). The server's input is an ℓ-bit value x which is encrypted under the Paillier cryptosystem $[x]$. Given $[x]$, f computes $f(\perp, [x]) = (\langle x \rangle_c, \langle x \rangle_s)$ which are secret shares of x for the client and the server.

Theorem 41. *The switching protocol π (Protocol 2) securely computes the functionality $f(\perp, [x]) = (\langle x \rangle_c, \langle x \rangle_s)$ in the presence of semi-honest, non-adaptive, computationally bounded adversaries.*

Proof. In the following, we prove Theorem 41 for a corrupted server and client separately, by showing that the view of adversary $\mathcal{A}$ in the real world is computationally indistinguishable from the simulated views of $\mathcal{S}_i$, where $i \in \{c, s\}$ is for the client and the server.

- **Server is corrupted by $\mathcal{A}$**: $\mathcal{S}_s$ is given the input and output of the server which are $[x], \langle x \rangle_s$, and the security parameter 1^κ. In simulation, we need to show that $\mathcal{S}_s$ can generate the view of incoming messages to the server, which is $\langle x + r \rangle_s$. $\mathcal{S}_s$ works as follows:

  1. $\mathcal{S}_s$ chooses a uniformly distributed random tape, r_1.
  2. $\mathcal{S}_s$ picks an $\ell + \kappa$-bit random value r' using the random tape r_1.
  3. $\mathcal{S}_s$ creates the secret shares $\langle r' \rangle_c$ and $\langle r' \rangle_s$.
  4. Using the output $\langle x \rangle_s$, $\mathcal{S}_s$ computes $\langle x + r' \rangle_s = \langle x \rangle_s + \langle r' \rangle_s$.

  The view of the server in the real world is

  $$\mathbf{view}_s^\pi(\perp, [x]) = ([x], r_s; \langle x + r \rangle_s), \tag{15}$$

  while the view generated by the simulator

  $$\mathcal{S}_s(1^\kappa, [x], \langle x \rangle_s) = \left([x], r_1; \left\langle x + r' \right\rangle_s\right). \tag{16}$$

  Since $\mathcal{S}_s$ does not have access to the decryption key sk, it cannot simulate $\text{decr}([x + r])$. On the other hand, it can generate the intermediary message $\langle x + r' \rangle_s$, but if r' is uniformly sampled from r_1, then

  $$\begin{aligned} \{\mathcal{S}_s(1^\kappa, [x], \langle x \rangle_s), f(\perp, [x])\} &\overset{\text{c}}{\equiv} \\ \{\mathbf{view}_s^\pi(\perp, [x]),\ \mathbf{output}^\pi(\langle x \rangle_c, \langle x \rangle_s)\}, \end{aligned} \tag{17}$$

  if for every nonuniform polynomial time distinguisher D there exists a negligible function $\mu(\kappa)$ such that

  $$\begin{aligned} \Big| &\Pr\left[D\left(\left([x], r_1; \left\langle x + r' \right\rangle_s\right) \wedge (\langle x \rangle_c, \langle x \rangle_s)\right) = 1\right] - \\ &\Pr\left[D\left(([x], r_s; \langle x + r \rangle_s) \wedge (\langle x \rangle_c, \langle x \rangle_s)\right) = 1\right] \Big| \leq \mu(\kappa). \end{aligned} \tag{18}$$

  Equation 18 holds due to the security of secure two-party computation and the uniformity of the random tape. The indistinguishability guarantees that a corrupted server has no advantage on differentiating $\langle x + r' \rangle_s$ from $\langle x + r \rangle_s$.

- **Client is corrupted by $\mathcal{A}$**: Different from the server, the client does not have an input for π. $\mathcal{S}_c$ is only provided the output $\langle x \rangle_c$ and the public and private keys pk, sk. To simulate the intermediary messages $[x + r]$ and $\langle r \rangle_c$, $\mathcal{S}_c$ works as follows:

  1. $\mathcal{S}_c$ chooses uniformly distributed random tapes r_1 and r_2.
  2. $\mathcal{S}_c$ picks an ℓ-bit random value x' and an $(\ell + \kappa)$-bit random value r' using the random tapes r_1, r_2.
  3. $\mathcal{S}_c$ encrypts $x' + r'$ as $[x' + r']$ using the public key pk.

4. $\mathcal{S}_c$ creates secret shares for r' such that $r' \rightarrow \langle r' \rangle_c + \langle r' \rangle_s$.

The view of the client in the real world and the view generated by the simulator are

$$\mathbf{view}_c^{\pi}(\perp, [x]) = (\perp, r_c; [x + r], \langle r \rangle_c), \tag{19}$$

$$\mathcal{S}_c(1^{\kappa}, \perp, \langle x \rangle_c) = \left(\perp, r_1, r_2; [x' + r'], \left\langle r' \right\rangle_c\right), \tag{20}$$

respectively. Then,

$$\{\mathcal{S}_c(1^{\kappa}, \perp, \langle x \rangle_c), f(\perp, [x])\} \stackrel{\text{c}}{\equiv} \{\mathbf{view}_c^{\pi}(\perp, [x]),\ \mathbf{output}^{\pi}(\langle x \rangle_c, \langle x \rangle_s)\} \tag{21}$$

in the existence of a negligible function $\mu(\kappa)$ for every nonuniform polynomial time distinguisher D such that

$$\Big|\Pr\left[D\left(\left(\perp, r_1, r_2; [x' + r'], \left\langle r' \right\rangle_c\right) \wedge (\langle x \rangle_c, \langle x \rangle_s)\right) = 1\right] - \Pr\left[D\left((\perp, r_c; [x + r], \langle r \rangle_c) \wedge (\langle x \rangle_c, \langle x \rangle_s)\right) = 1\right]\Big| \leq \mu(\kappa). \tag{22}$$

Equation 22 is correct when a semantically secure encryption scheme and secret sharing scheme are used in securing messages $[x + r]$ and $\langle r \rangle_c$ which eliminate the advantage of distinguishing $[x + r]$ from $[x' + r']$ and $\langle r \rangle_c$ from $\langle r' \rangle_c$. Using the Paillier encryption scheme, which satisfies the semantic security under the decisional composite residuosity assumption, and arithmetic secret sharing, which guarantees information theoretic security, a corrupted client cannot break the indistinguishability. Furthermore, the adversary cannot reveal any information about x from the decryption of $[x + r]$, given that a sufficiently large, uniformly random value ($\ell + \kappa$ bits) is selected for masking x.

□

5 Performance Evaluation

We implemented SwaNN to evaluate its performance in different settings and to compare it with the state-of-the-art. We used the C++ programming language for the implementation and GMP 6.1.2 library for big integer operations. We used the ABY framework [19] for secure two-party computation operations. For the homomorphic operations, we used the Paillier implementation of ABY due to its efficiency. We selected 2048 bits modulus size in Paillier operations to meet the current security standards. For the ABY operations we selected 32-bit shares. The machine we used in the experiments runs Ubuntu 16.04 operating system with Intel Core i5-3470 CPU@3.20GHz.

5.1 Optimizing Computations

In each layer of neural networks, the same operations are repeated for each index of the input independently. Thus, in our implementation we use several optimization techniques which help reduce the computation time and communication usage by enabling simultaneous execution. To optimize 2PC computations, we use single instruction multiple data (SIMD) techniques [41] which are provided in the ABY framework. SIMD techniques cannot be fully utilized for the computations with the Paillier cryptosystem. Therefore, to improve the efficiency in homomorphic encryption, we adapt two techniques to the Paillier encryption which enables simultaneous computation.

The first technique we use is data packing. It packs multiple data items into a single ciphertext as described in [42]. Accordingly, we create slots of $t + \kappa$ bits for each data item where κ is the security parameter and t is the length of the data item. Given the plaintext modulus N, we can pack $\rho = \left\lfloor \frac{\log_2 N}{t+\kappa} \right\rfloor$ items in a single ciphertext as in Equation 23.

$$[\hat{x}] = \sum_{m=0}^{\rho-1} [x_{i,j}] \cdot (2^{t+\kappa})^m \tag{23}$$

Using data packing we can use the full plaintext domain in the Paillier cryptosystem and perform additions on the packed ciphertext simultaneously. Furthermore, in interactive protocols, using data packing helps reduce the bandwidth usage and the cost of decryption operations.

The second technique we use to improve efficiency of homomorphic encryption is using a multi-exponentiation algorithm to simultaneously perform the operations in the form of

$$\prod_{i=1}^{w} a_i^{b_i} = a_1^{b_1} \cdot a_2^{b_2} \dots a_w^{b_w}. \tag{24}$$

Lim-Lee's multi-exponentiation algorithm [43,44] enables to perform Equation 24 simultaneously by modifying the binary exponentiation algorithm using several precomputation techniques. In our work, we can apply multi-exponentiation for the computation of dot product (Equation 9) over encrypted data thanks to the additive homomorphism of the Paillier cryptosystem. We summarize the optimizations used in each layer of neural networks as follows:

- **`Conv`**: Multi-exponentiation technique is used to reduce the cost of dot products.
- **`Act`**: Data packing is used before performing the activation function. If activation is performed with 2PC operations, then SIMD optimization is used.
- **`Pool`**: No optimization technique is needed.
- **`FC`**: Multi-exponentiation technique is used to reduce the cost of matrix multiplications.

5.2 Experiments

Table 2: Computation time per layer in the client-server and the two-server scenario (in ms). The timings are provided for optimized and non-optimized PHE-only setting and optimized hybrid setting. The total timings marked with * show the simultaneous run time of SwaNN for two images.

	Non-optimized - PHE only				Optimized - PHE only				Optimized - Hybrid			
Layer	Client	Server	Server-1	Server-2	Client	Server	Server-1	Server-2	Client	Server	Server-1	Server-2
`Conv`	–	1873	1850	1857	–	372	377	369	–	371	370	372
`Act`	12629	15754	32680	32536	2484	19038	23494	23328	2366	3714	6078	6164
`Pool`	–	35	32	32	–	34	32	32	–	33	32	32
`Conv`	–	2852	2843	2831	–	550	566	547	–	548	548	550
`Pool`	–	35	35	35	–	35	37	35	–	35	35	35
`FC`	–	6395	6333	6319	–	2238	2227	2211	–	2216	2219	2234
`Act`	1496	1864	3846	3847	311	2207	2758	2758	273	501	798	786
`FC`	–	9	9	9	–	9	9	9	–	9	9	9
Total	42915		47466*		27248		29500*		10066		10182*	

We design two experiments with respect to the activation function used in the neural network. In the first experiment we used x^2 as the activation function and re-trained the neural network structure used in CryptoNets [8]. In the second experiment we used ReLU as the activation function and re-trained the neural network structure used in MiniONN [13]. The properties of the neural networks are detailed in Appendix A.

5.2.1 Experiment 1

In the first experiment, we measured the performance of SwaNN with x^2 activation function in the client-server and the two-server scenario. For each scenario, we designed two different cryptographic setting. The first setting is an **only-PHE** setting which is totally based on the Paillier cryptosystem. We implemented the activation function x^2 as described in Protocol 1. The second setting is a **hybrid setting** where the computation switches between PHE and 2PC. We implemented the secure switching protocols in Protocol 2 and Protocol 3 for this setting and implemented x^2 using the ABY framework. Table 2 demonstrates the performance of SwaNN for both scenarios in the only-PHE and the hybrid setting for each layer of the network. For the only-PHE setting we provide the timings with and without optimizations. For the hybrid setting, we provide only optimized timing values.

The results show that in the client-server scenario when no optimizations are used, the prediction of one image is computed approximately in 43 seconds. However, when we use optimization techniques, we can reduce the computation time to 27 seconds. In a hybrid setting, this cost is reduced to 10 seconds. Furthermore, in the two-server scenario with a slight increase in computation time, two images can be processed simultaneously. More particularly in an optimized hybrid setting the two servers can compute the prediction result for two images in 10 seconds simultaneously.

Table 3: Detailed computation time for the activation layer in the client-server and the two-server scenario for the hybrid setting (in ms).

Operation	Client	Server	Server-1	Server-2
Packing	–	3544	3551	3553
Decryption	73	–	142	146
Unpacking	0.1	–	0.1	
ABY	11	14	27	27
Encryption	2282	156	2358	2437
Total	6069		6078*	

In Table 3 we provide the details of the computation time for the activation layer in the hybrid setting. The packing, decryption and unpacking operations are performed during the switching from PHE to 2PC. The encryptions are computed by both parties when switching the operations from 2PC to PHE. In the client-server scenario, the client spends 2.36 seconds for the computations while the server spends approximately 3.7 seconds. In the two-server scenario, both servers spend approximately 6 seconds for the computation of the activation layer of two images.

Apart from computation time, we also analyzed the bandwidth usage of SwaNN for different settings. Table 4 shows the communication cost in both scenarios for the only-PHE setting and the hybrid setting. The packing technique used in the activation layers helps reduce the bandwidth usage by half. Besides due to the interactive nature of 2PC, the bandwidth usage in the hybrid setting is higher than the only-PHE setting for both scenarios.

Table 4: Bandwidth usage of SwaNN in different settings (in MB).

	Client-Server	Two-Server
PHE only (w/o opt.)	0.97	0.96
PHE only (w/ opt.)	0.51	0.51
Hybrid (w/ opt.)	1.63	1.73

As a final analysis, in Table 5 we compare SwaNN with the state-of-the-art works CryptoNets [8] and MiniONN [13] with respect to computation time and bandwidth usage. CryptoNets, which uses fully homomorphic encryption for computations, requires 297.5 seconds for one prediction. The protocol enables simultaneous computation by packing 4096 images into a single ciphertext. This is an advantage when the same client has very large number of prediction requests. MiniONN can compute the prediction result for the same network in 1.28 seconds. However, this computation requires 47.6 MB bandwidth usage. SwaNN can com-

pute the same prediction result in 10 seconds. Although the computation time of SwaNN is higher than MiniONN, SwaNN achieves a 27-fold less bandwidth usage.

Table 5: Comparison with the state-of-the-art in Experiment 1.

	Computation time (s)	Bandwidth usage (MB)
CryptoNets [8]	297.5	372.2
MiniONN [13]	1.28	47.6
SwaNN	10.1	1.73

5.2.2 Experiment 2

As the second experiment, we measured the performance of SwaNN with ReLU activation function for the network described in Table 9 in Appendix A. We used maximum operation for pooling layers. We provide the timings for the max pooling along with ReLU function since we implemented them together. We measure the timings in the client-server and the two-server scenario only with optimizations. Table 6 details the computation time for each layer. Due to larger number of input size in each layer of the network, the computation cost of SwaNN reaches to 61 seconds. The first activation layer is the dominant layer in the run time. As expected, the high computation cost is caused by the decryption operations which are performed during the switching phase from PHE to 2PC.

Table 6: Computation time per layer in the client-server and the two-server scenario (in ms).

Layer	Client	Server	Server-1	Server-2
`Conv`	–	4118	4102	4098
`Act+Pool`	6855	46795	48777	48869
`Conv`	–	460	457	457
`Act+Pool`	766	4418	5602	5597
`FC`	–	1318	1329	1331
`Act`	277	506	815	815
`FC`	–	6	6	6
Total	57824		61173	

In Table 7, we compare the performance of SwaNN with MiniONN. Clearly, MiniONN outperforms SwaNN almost 7-fold in computation time. However, in terms of communication, SwaNN is more efficient with a bandwidth usage of 228 MB (compared to 657 MB in MiniONN).

Table 7: Comparison with the state-of-the-art in Experiment 2.

	Computation time (s)	Bandwidth usage (MB)
MiniONN [13]	9.32	657.5
SwaNN	61.17	228.1

6 Conclusion

We have proposed a privacy preserving neural network prediction protocol that combines the additively homomorphic Paillier encryption scheme with secure two-party computation. Thanks to the use of the Paillier encryption algorithm for linear operations and also the x^2 activation function, the solution achieves better computational cost compared to existing HE-based solutions. Different computation optimisations based on the use of data packing and the multi-exponentiation algorithm have been implemented. Furthermore, the communication cost is also minimized since 2PC is only used for non-linear operations (max pooling and/or RELU). SwaNN can be executed in the two-server setting, in case the client lacks resources. Experimental results show that SwaNN actually achieves the best of both worlds, namely, better computational overhead compared to HE-based solutions and, better communication overhead compared to 2PC-based solutions.

References

[1] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptology ePrint Archive*, vol. 2017, p. 35, 2017. [Online]. Available: http://eprint.iacr.org/2017/035

[2] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 826–834, 1983.

[3] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CoRR*, vol. abs/1202.2745, p. 20, 2012. [Online]. Available: http://arxiv.org/abs/1202.2745

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012, pp. 1106–1114.

[5] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision*

and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. IEEE Computer Society, 2015, pp. 815–823.

[6] N. Jones, "Computer science: The learning machines," *Nature: Computer Science*, vol. 505, no. 7482, pp. 146–148, January 2014. [Online]. Available: https://www.nature.com/news/computer-science-the-learning-machines-1.14481

[7] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "Mlaas: Machine learning as a service," in *14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11, 2015*. IEEE, 2015, pp. 896–902.

[8] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. JMLR.org, 2016, pp. 201–210.

[9] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proceedings of the 8th workshop on Multimedia & Security, MM&Sec 2006, Geneva, Switzerland, September 26-27, 2006*. ACM, 2006, pp. 146–151.

[10] C. Orlandi, A. Piva, and M. Barni, "Oblivious neural network computing via homomorphic encryption," *EURASIP J. Information Security*, vol. 2007, pp. 1–11, 2007.

[11] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 19–38.

[12] P. Mohassel and P. Rindal, "Aby^3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018, pp. 35–52.

[13] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 619–631.

[14] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 2018, pp. 1651–1669.

[15] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. Springer, 1999, pp. 223–238.

[16] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*. Springer, 2001, pp. 119–136.

[17] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. Springer, 1991, pp. 420–432.

[18] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. ACM, 1987, pp. 218–229.

[19] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015, pp. 1–15.

[20] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*. ACM, 2018, pp. 2:1–2:6.

[21] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*. ACM, 2018, pp. 707–721.

[22] M. Dahl, J. Mancuso, Y. Dupis, B. Decoste, M. Giraud, I. Livingstone, J. Patriquin, and G. Uhma, "Private machine learning in tensorflow using secure computation," *CoRR*, vol. abs/1810.08130, 2018. [Online]. Available: http://arxiv.org/abs/1810.08130

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan,

F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[24] S. Wagh, D. Gupta, and N. Chandran, "Securenn: Efficient and private neural network training," in *Privacy Enhancing Technologies Symposium*. (PETS 2019), February 2019.

[25] "Simple Encrypted Arithmetic Library (release 3.1.0)," Dec. 2018, microsoft Research, Redmond, WA. [Online]. Available: https://github.com/Microsoft/SEAL

[26] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *CoRR*, vol. abs/1711.05189, 2017. [Online]. Available: http://arxiv.org/abs/1711.05189

[27] A. Ibarrondo and M. Önen, "Fhe-compatible batch normalization for privacy preserving deep learning," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*. Springer, 2018, pp. 389–404.

[28] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*. Springer, 2018, pp. 483–512.

[29] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachéne, "Tfhe: Fast fully homomorphic encryption over the torus," Cryptology ePrint Archive, Report 2018/421, 2018, https://eprint.iacr.org/2018/421.

[30] A. Sanyal, M. J. Kusner, A. Gascón, and V. Kanade, "TAPAS: tricks to accelerate (encrypted) prediction as a service," *CoRR*, vol. abs/1806.03461, 2018. [Online]. Available: http://arxiv.org/abs/1806.03461

[31] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *PoPETs*, vol. 2018, no. 3, pp. 123–142, 2018.

[32] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018, pp. 1209–1222.

[33] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *CoRR*, vol. abs/1811.09953, 2018. [Online]. Available: http://arxiv.org/abs/1811.09953

[34] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 619–636.

[35] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *CoRR*, vol. abs/1803.05961, 2018. [Online]. Available: http://arxiv.org/abs/1803.05961

[36] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. [Online]. Available: https://openreview.net/forum?id=rJVorjCcKQ

[37] T. Toft, "Sub-linear, secure comparison with two non-colluding parties," in *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*. Springer, 2011, pp. 174–191.

[38] A. C. Yao, "Protocols for secure computations (extended abstract)," in *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 1982, pp. 160–164.

[39] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: tool for automating secure two-party computations," in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. ACM, 2010, pp. 451–462.

[40] Y. Lindell, "How to simulate it - A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography.* Springer International Publishing, 2017, pp. 277–346.

[41] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

[42] T. Bianchi, A. Piva, and M. Barni, "Composite signal representation for fast and storage-efficient processing of encrypted signals," *IEEE Trans. Information Forensics and Security*, vol. 5, no. 1, pp. 180–187, 2010.

[43] C. H. Lim and P. J. Lee, "More flexible exponentiation with precomputation," in *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*. Springer, 1994, pp. 95–107.

[44] C. H. Lim, "Efficient multi-exponentiation and application to batch verification of digital signatures," *Unpublished manuscript, August*, vol. n.a., no. n.a., p. 1, 2000.

A Neural Network Structures

In our experiments, we use two neural network structures which are previously trained by CryptoNets [8] and MiniONN [13] to perform image classification on MNIST data. Table 8 summarizes the structure of the neural network proposed in CryptoNets. The accuracy of the networks is 98.95%. The network has 9 layers. Since the last layer of the network, the sigmoid activation, is applied only in the training phase, we did not include it in our experiments. The activation function of the network is x^2. As pooling operation, scaled mean pooling is used. Secondly, we used the neural network structure proposed in MiniONN (Figure 12) [13]. The accuracy of the network is 99.31%. Table 9 demonstrates the layers of the network. The activation function of the network is ReLU. Max pooling is used in the pooling layer.

Table 8: CryptoNets Neural Network structure [8].

Layer	Input size	Output size	Filter	Stride
Conv	28×28	$5 \times 13 \times 13$	5×5	(2,2)
Act	$5 \times 13 \times 13$	$5 \times 13 \times 13$		
Pool	$5 \times 13 \times 13$	$5 \times 13 \times 13$	3×3	1
Conv	$5 \times 13 \times 13$	$50 \times 5 \times 5$	5×5	(2,2)
Pool	$50 \times 5 \times 5$	$50 \times 5 \times 5$	3×3	1
FC	$50 \times 5 \times 5$	100×1		
Act	100×1	100×1		
FC	100×1	10×1		

Table 9: MiniONN Neural Network structure [13].

Layer	Input size	Output size	Filter	Stride
Conv	28×28	$16 \times 24 \times 24$	5×5	(1,1)
Act	$16 \times 24 \times 24$	$16 \times 24 \times 24$		
Pool	$16 \times 24 \times 24$	$16 \times 12 \times 12$	2×2	2
Conv	$16 \times 12 \times 12$	$16 \times 8 \times 8$	5×5	(1,1)
Act	$16 \times 8 \times 8$	$16 \times 8 \times 8$		
Pool	$16 \times 8 \times 8$	$16 \times 4 \times 4$	2×2	2
FC	$16 \times 4 \times 4$	100×1		
Act	100×1	100×1		
FC	100×1	10×1		